Laukkanen, Eero; Mäntylä, Mika

Survey reproduction of defect reporting in industrial software development

Published in:
5th International Symposium on Empirical Software Engineering and Measurement (ESEM 2011), Banff, Alberta, Canada, September 19-23, 2011

DOI:
10.1109/ESEM.2011.28

Published: 01/01/2011

Document Version
Peer reviewed version

Survey Reproduction of Defect Reporting in Industrial Software Development

Eero I. Laukkanen
Aalto University, SoberIT
P.O. Box 19210,
FI-00076 Aalto, Finland
eero.laukkanen@aalto.fi

Mika V. Mäntylä
Aalto University, SoberIT
P.O. Box 19210,
FI-00076 Aalto, Finland
mika.mantyla@aalto.fi

***Abstract*—Context: Defect reporting is an important part of software development in-vivo, but previous work from open source context suggests that defect reports often have insufficient information for defect fixing. Objective: Our goal was to reproduce and partially replicate one of those open source studies in industrial context to see how well the results could be generalized. Method: We surveyed developers from six industrial software development organizations about the defect report information, from three viewpoints: concerning quality, usefulness and automation possibilities of the information. Seventy-four developers out of 142 completed our survey. Results: Our reproduction confirms the results of the prior study in that "steps to reproduce" and "observed behaviour" are highly important defect information. Our results extend the results of the prior study as we found that "part of the application", "configuration of the application", and "operating data" are also highly important, but they were not surveyed in the prior study. Finally, we classified defect information as "critical problems", "solutions", "boosters", and "essentials" based on the survey answers. Conclusion: The quality of defect reports is a problem in the software industry as well as in the open source community. Thus, we suggest that a part of the defect reporting should be automated since many of the defect reporters lack technical knowledge or interest to produce high-quality defect reports.**

Keywords-software debugging; software maintenance; software quality;

I. Introduction

Defect reporting is a vital part of software engineering, but its current state of the practice is far from ideal. This can be seen in prior work by Zimmerman et al. [1] who state, based on studies of open source projects, that defect reports often have insufficient or incorrect information, which increases the effort needed for defect fixing. Our industrial collaboration and research have revealed that the same problems exist in the software industry [2]. We see the problem of inadequate defect information rising from two sources: insufficient skills of defect reporters and the manual workload of defect reporting.

First, not every defect reporter knows what information developers consider helpful or need for fixing defects. The role and background of a defect reporter can vary between two extremes [2]; the reporters can be programmers with highly technical background and in-depth knowledge of the software. However, they can also be sales representatives, who may possess expertise in using the system, but have very little knowledge of or even interest in the technical details of the software. Sales representatives and other domain experts can find defects with high business value when preparing for a customer demonstration using an internal alpha version of the upcoming release [2]. Overlooking their bug reports could be disastrous for a software company.

Second, if too much information is required in manual defect reporting there is a risk that the reporter will not submit the defect report at all. This reaction is similar to survey fatigue where a respondent will fail to complete a lengthy survey. Making all fields in bug reporting system mandatory will decrease the number of defect reports. Furthermore, one should also understand that time spent on defect reporting is separate from the defect reporters real job: programming or selling the software.

Replication and reproduction has been recognized throughout the natural, social and the engineering sciences as ways of creating and deepening scientific knowledge. In software engineering research the initial focus on replications was given for replicating experiments [3]–[5]. However, in recent years there has been a growing interest in case studies, surveys, and other field studies in the software engineering research community [6]–[9]. This has sparked interest in broader view of replications [10], [11] that acknowledges the need to replicate such studies.

We surveyed defect reporting in six industrial software organizations. Our study was a reproduction of prior work by Zimmerman et al. [1]. We studied the defect information from three viewpoints. First, we wanted to know what defect information is useful for fixing defects. Second, we wanted to find out what defect information is missing or incorrect in the studied organizations. Third, we wanted to know which items should be collected in the defect reports.

Next, we present related work to our study (Section II) and describe our research methodology and the survey design (Section III). We present the results from the survey (Section IV) and compare them to the original study [1] we reproduced (Section V). Finally, we present our conclusions and propose future work (Section VI).

II. Related work

Zimmermann et al. [1] defined the characteristics of a good defect report. They conducted a survey of software developers and defect reporters of three open source software projects, to find out what information developers had used for defect fixing and what information developers consider the most helpful for fixing defects. *Steps to reproduce, stack traces, test cases, screenshots, observed behavior*, and *expected behavior* were

rated as the most important, but also the hardest for reporters to provide. Some of the items that developers consider helpful were not reported as frequently as might be assumed. In addition to the study by Zimmermann et al., many defect reporting guidelines in the Internet offer useful information for defect reporting, such as "How to Report Bugs Effectively" by Tatham [12].

Our previous work [2] examined testing activities in different industrial case companies. In all companies testing was conducted by multiple stakeholders, not by the specialized testers alone. Companies valued domain knowledge and the viewpoint of end-users, which led to collaboration in testing. We also noted that the state of defect databases in companies was a concern, since the database entries were not comprehensive. A similar observation was made by Aranda and Venolia [13]. They suggest that inspecting the electronic imprints of defects in defect databases is not enough to understand the history of a defect. Instead, it is required to communicate with the people related to the fixing process. Breu et al. [14] searched defect reports, extracted and categorized the questions asked in the reports. They found that 15.1% of all the questions were about missing information, which shows problems with defect reporting.

Merkel and Kanij [15] conducted a global online survey of what testers believed the characteristics of a good tester. They found that testing performance varies among testers. The quality of a bug report was the most important factor in a tester with high performance. Expertise in the problem domain was agreed as a factor that influences the performance of software testers, in addition to knowledge of specific testing techniques.

To make defect reporting easier for end-users, multiple automatic reporting tools have been implemented for desktop environments, such as Windows Error Reporting [16] for Microsoft Windows. Some important information can be collected with sophisticated methods that are not used by current automatic reporting tools. Artzi et al. [17] propose a method of collecting test cases automatically although with 13%–64% performance overhead. Castro et al. [18] show how the input that causes the defect can be collected without collecting private information. Zamfir et al. [19] introduce defect fingerprints that would assist with the fixing of concurrency defects.

III. Method

This section describes how our study was conducted. We begin by listing the research questions. Next, we explain the selection of the participants for our survey. Then, we present the survey instrument we used. We explain our data analysis methods and conclude by describing how our study was replicated that of Zimmermann et al. [1].

A. Research Questions

To understand the problems of defect reporting, we defined our research questions as follows:

1) *What defect information do developers consider useful for fixing defects?*
2) *Has the useful information been incorrect or missing in defect reports?*
3) *Which information should automatically be collected into defect reports?*

Answers to the first question tell us about the kind of defect data that should be included in the defect reports. Answers to the second question tell us how problematic the reporting of a particular piece of defect information is, thus giving us a hint of the possible impact of automatic collection. Finally, answers to the third question classify the items based on whether they can be collected automatically or they must be collected by other means.

B. Selection of Participants

Our five participant companies (Table I) were selected based on research cooperation within our research project. Our participant companies did software development in multiple contexts and every company had several software projects. Thus, multiple areas of software development have been included in the survey population. This was an important factor because our first research question could depend on the context of software development.

Developers were contacted by email with LimeSurvey survey application [20]. One hundred and forty two developers were invited to the survey and 74 completed the survey (Table I). The response rate was 52.1% which is comparable to that of other online surveys [21]. We believe that response rate increased thanks to research cooperation with case companies and personal reminders during the administration of the survey.

Based on the first section on the questionnaire we got additional information on the population of participants. All participants were experienced developers. Half of them had worked in their company for more than three years and as a developer for more than six years. Everyone had worked at least one year as a developer. Seventy-three percent had fixed 100 or more defects in the company. Sixteen participants had subordinates in their work and 58 did not.

Defect reporters in our participant companies were defined in our prior work [2] in which we categorized defect reporters based on how closely they worked with the customers. Defect reporters are not a homogeneous group; they range from external customers who use the software for their own intentions, to software developers.

C. Questionnaire Design

We asked software developers working in five selected case companies to complete a half-open survey. A half-open survey contains both closed and open questions, to obtain quantitative and qualitative data from the respondents. The questionnaire was based on the survey by Zimmermann et al. [1] and other survey literature [21]–[23]. It was also piloted by two researchers inside authors' research group to enhance the reliability of the questionnaire. The questionnaire was split into six pages:

Page 1. **Background information.** Participants were asked how experienced they were as developers and

TABLE I
SUMMARY OF THE CASE COMPANIES. TWO BRANCHES WERE SEPARATED IN THE ANSWERS FROM COMPANY B. THE ACCURACY OF PERSONNEL IS 50 PERSONS AND THE ACCURACY OF AGE IS 10 YEARS.

Organization	Personnel	Age (years)	Domain	Invited	Responses	Completed
A	50	30	Automation systems & Software system for operation	17	8 (47.1%)	6 (35.3%)
B1	500	40	COTS for engineering design	31	18 (58.1%)	12 (38.7%)
B2	500	40	Software systems for operation and engineering design	18	10 (55.6%)	9 (50.0%)
C	100	10	Internet applications for administration	23	20 (87.0%)	18 (78.3%)
D	100	20	Business software system of specific industry	31	21 (67.7%)	19 (61.3%)
E	100	20	COTS for engineering design & Software system for operation	22	11 (50.0%)	10 (45.5%)
Total				142	88 (62.7%)	74 (52.1%)

how many defects they had fixed. They were also asked whether or not they had subordinates.

Page 2. **Defect information.** Participants were asked to select useful pieces of information (later called *items*) for fixing defects. The items were on a preselected list, but respondents could also write their own items in separate text boxes. The respondents were asked to select items even if those items would be only slightly useful.

Page 3. **Defect information usefulness.** The items participants selected on the previous page were rated on their usefulness along a 5-point scale: "Not useful at all," "Slightly useful," "Fairly useful," "Quite useful," "Very useful." The items which were not selected in the previous section were not rated, to make the questionnaire consistent to the participant. Respondents could select an item to be useful on Page 2, but then rate it "Not useful at all." Therefore all answers where option "Not useful at all" was selected were discarded as inconsistent.

Page 4. **Missing or incorrect information in defect reports.** The items participants selected on Page 2 were rated to be missing or incorrect with a 5-point scale: "Never or very seldom," "Quite seldom," "Sometimes," "Quite often," "Very often or always."

Page 5. **Collecting information automatically.** The items participants selected on Page 2 were rated how easily they could be automatically collectible with a 5-point scale: "Very hard," "Quite hard," "I don't know," "Quite easy," "Very easy." Participants could also elaborate on how these items could be collected.

Page 6. **Comments.** Finally, participants could leave extra comments.

On page 2, participants could select useful items from a preselected list of 18 items (Table III). The list was based on the list used by Zimmermann et al. [1], but modified under our consideration and survey piloting. We also allowed respondents to optionally add a maximum of three useful items to the survey.

D. Analyzing the Results

We analyzed our results by examining the distribution of all answers instead of only calculating medians. To analyze the results, we calculated combinations of different factors, for example, how many respondents selected an item to be both useful and missing or incorrect. By merging two factors of an item we could determine if an item causes a problem in defect reporting that can be solved. When calculating the combinations, developers were handled one at a time to avoid false conclusions. For example, one person could rate an item to be useful and another could rate that item as missing or incorrect. That case would not have been counted as a combination.

After analyzing the quantitative results, we read the comments given by participants on page 5 and on page 6 of the questionnaire. Comments could support quantitative results and give us a better understanding of defect reporting in the companies.

E. Replication of a Previous Study

Gómez et al. [11] show that replications are not classified in any standardized manner in the sciences. In the field of experimental software engineering our work should be viewed as reproduction [24] to differentiate it from replications that are mostly understood in software engineering as exact replications (Table II). For this reason and because replications in software engineering focus mostly on experiments, we use the terminology of Tsang and Kwan [25] to describe our survey replication. They classified replications into six types along two dimensions (Table II). The first dimension describes whether or not the same measurement and analysis methods are used. The second dimension depends on the source of the data. In this study, measurement and analysis methods and the target population were different from those in the original study [1]. Therefore we classified our replication as a generalization and extension.

TABLE II
TYPES OF REPLICATION BY TSANG AND KWAN [25]

	Same Measurement and Analysis	**Different Measurement and Analysis**
Same Data Set	Checking of analysis	Reanalysis of data
Same Population	Exact replication	Conceptual extension
Different Population	Empirical generalization	Generalization and extension

Zimmermann et al. [1] also surveyed the useful information for defect fixing and problems of defect reporting. Our study

should be seen as a reproduction [24] of that. Our study differs from theirs in the following respects:

- Our survey population consisted of industrial software developers and their survey population was open source software developers. The difference is the context of software development. In open source, software development is open, meaning that everyone can contribute to the project by developing or testing the software. Industrial software development, at least in our participant companies, is closed, meaning that only a certain population has access to the development process. Of course, it is possible that open source software can be developed in the industry.
- We added *part of the application, configuration of the application, operating data, reporter's contact information*, and *user input* to the list of defect information. *Configuration of the application* and *operating data* were added to our survey as we knew our companies had products with several thousand configuration points and where operating data was crucial. Thus, we had prior insight that bugs could often result from different configuration and operating data [26]. Adding *part of the application* and *user input* was done, mostly based on our experiences.
- We combined *build information* and *version information* as part of *product information*. We renamed *summary* to *bug title* as it was called in our case companies. We also considered that *code examples* will be covered in either *steps to reproduce*, *test scripts* or *user input* fields.
- In the survey by Zimmermann et al. developers did not rate items. Instead, they selected the three most useful items. We introduced rating, based on that there could be more than three useful items for fixing defects.
- We studied the problems related to defect reporting by rating individual items. Zimmermann et al. considered other possible problems such as *bad grammar*. Our approach was better suited to our focus of automatic defect reporting.
- Zimmermann et al. asked the opinions of defect reporters. We instead used a section (page 5 in the questionnaire) about automatic collection of items because it was the focus of our study.

Our survey reproduction differs from the original study on two dimensions. First, our survey population consisted of industrial software developers; in the previous study the population was open source software developers. Second, our survey design had a different software development context. By comparing the results of both studies we can ascertain which qualities of open source software development apply to industrial software development. A possible threat to validity is introduced, because we had multiple differences from previous study. If our results differ, we will not always know why. Thus, the main contribution of our study is to strengthen previous results, and conflicts between the results should be discussed with caution.

IV. Results

Next we introduce the quantitative results from the survey. From each quantitative section on the questionnaire, we present the three highest-rated items and the three lowest- or least-rated items. We also discuss our other observations from the results. Finally, we combine different results to see which items should be collected automatically.

A. Useful Information

Developers were to select items they considered **useful pieces of information for fixing defects**. Out of 18 items, the mean count of selected items was 12.65 with the standard deviation of 3.69. Every developer selected at least six items.

Three items were selected by over 90% of the respondents: *steps to reproduce* (97%), *screenshots* (95%), and *part of the application* (92%) (Table III). The selection of these items is not surprising, because these items can be useful for the majority of defects. *Steps to reproduce* is a vital piece of information for understanding a defect. If a developer cannot reproduce the defect, he or she can seldom resolve it. Three **least selected items** were: *hardware context* (34%), *test cases and test scripts* (47%), and *severity of the bug* (54%). *Hardware context* was not selected probably because the target companies of the survey were not developing software close to the hardware. The situation could be different if device drivers were developed, for example. Most developers did not select *test cases and test scripts*, which is somewhat surprising. One explanation can be that detailed *test cases* are not always used in our target companies. *Severity of the bug* can be used for bug triaging, but it is not useful after the defect to be fixed has been identified.

TABLE III
Selection% and Usefulness of Items

Item	Selected	Slightly Useful	Fairly Useful	Quite Useful	Very Useful
Bug title	64%	23%	34%	26%	17%
Component / module	77%	12%	16%	40%	32%
Configuration	82%	7%	31%	36%	26%
Error reports	70%	13%	17%	38%	31%
Expected behavior	69%	2%	18%	35%	45%
Hardware context	34%	40%	44%	8%	8%
Observed behavior	77%	5%	7%	28%	60%
Operating data	89%	6%	20%	26%	48%
Part of the application	92%	3%	6%	25%	66%
Product information	64%	13%	30%	26%	32%
Contact information	58%	33%	30%	19%	19%
Screenshots	95%	4%	19%	27%	50%
Severity of the bug	54%	30%	45%	22%	2%
Software context	57%	31%	40%	21%	7%
Stack trace	70%	8%	17%	35%	40%
Steps to reproduce	97%	0%	0%	3%	97%
Test cases, test scripts	47%	9%	40%	26%	26%
User input	69%	4%	22%	51%	24%

Selection counts show that **developers were not unanimous in their selections** (Table III). For example, *test cases, test scripts* were selected by 47% of the developers and not

selected by 53%. The same observation was made inside the companies, for example inside Company C 33% of the developers selected *test cases, test scripts*. This could mean that either the developers are not used to using all of the items for fixing defects, or some items are more useful than others for fixing certain defects. The latter is clearly the case with *hardware context*, since it could be a fundamental piece of information for fixing certain defects.

Items which were most frequently rated to be **very useful for fixing defects** were: *steps to reproduce* (97%), *part of the application* (66%), and *observed behavior* (60%) (Table III). *Steps to reproduce* is clearly the most selected item and the most useful. The only difference here to the selection count of the items was that *screenshots* were not considered "Very useful" as much as *observed behavior* by those who selected the items in the first place. While *screenshots* are generally useful information for fixing defects, they are not useful for all kinds of defects and *observed behavior* can describe all defects much more diversely. Three **items not considered very useful for fixing defects** were *severity of the bug* (2%), *software context* (7%), and *hardware context* (8%). The result is to be expected in light of the previous result. Here *test cases and test scripts* stand up compared with the previous result. We propose that those who consider *test cases and test scripts* useful are accustomed to using them for fixing defects. Our last observation is that *expected behavior* is rated quite low. This is surprising because it has been a problem when the reporter and the developer have different understandings of the defect.

Differences among companies were explored by comparing answers from the Companies C and D, because they both had a high response rate and their domains were distinct. Company C does Internet applications and Company D does data systems and simulators. However, no clear differences were found between companies concerning useful pieces of information for defect fixing. When comparing selection counts between companies, the largest difference was with *user input* (30% difference between selection counts). We think there are no significant differences because our preselected list contained pieces of information which can be used for fixing defects in various software development contexts. In addition, in both companies there were multiple software projects and thus, diverse software development activities.

Respondent could also type in **additional items which they consider useful for fixing defects**. Fourteen participants (19%) submitted total of 22 additional items. Out of those, three items were in the list we already provided in the survey, but they were more detailed, *video capture of how to reproduce*, for example. Four submitted items were information that could be acquired from the reporter and other 13 items were mainly related to developers' software context (e.g. web application). This result indicates that some useful information for fixing defects is connected tightly to the operating environment of the application. It highlights the importance of software context in defect reporting. Thus, it can not be determined universally what information should be included into defect reports. Two given additional items were discarded, because they could be understood in multiple ways. Since only 19% of the participants provided us with additional items and since those additional items were heavily diversified, we did not analyze quantitative answers regarding additional pieces of information. We believe that our preselected list of useful items was comprehensive enough for typical defects.

B. Missing or Incorrect Information

Developers rated how often an item had been missing or incorrect in the defect reports. Items which were most frequently rated to be **very often missing or incorrect in defect reports** were: *hardware context* (40%), *stack trace* (35%), and *software context* (24%) (Table IV). Both *hardware context* and *software context* were rated rather low in the previous section (Table III). A reporter can have the mistaken assumption that the problem exists in the used software when the defect can be caused by hardware or concurrent software. Either way, developers who need these items find the items to be missing or incorrect quite often. Remember that only developers who considered these items useful in the first place answered in this section. *Stack trace* was missing quite often according to 54% of respondents. In the viewpoint of a non-technical computer user, *stack traces* might be only loosely related to the defect itself. Other items which were notably rated as missing or incorrect quite often were *steps to reproduce* (64%) and *screenshots* (50%). Both were rated high in the previous section, which means that they are useful but are not reported correctly.

TABLE IV
LACK OF ITEMS

Item	Never or Very Seldom	Quite Seldom	Sometimes	Quite Often	Very Often or Always
Bug title	32%	32%	28%	6%	2%
Component / module	5%	33%	33%	26%	2%
Configuration	3%	16%	21%	38%	21%
Error reports	13%	17%	40%	17%	12%
Expected behavior	2%	12%	29%	51%	6%
Hardware context	8%	20%	12%	20%	40%
Observed behavior	4%	33%	44%	16%	4%
Operating data	5%	18%	44%	23%	11%
Part of the application	12%	31%	43%	12%	3%
Product information	15%	26%	40%	15%	4%
Contact information	47%	23%	19%	9%	2%
Screenshots	10%	11%	29%	47%	3%
Severity of the bug	12%	32%	38%	12%	5%
Software context	7%	19%	14%	36%	24%
Stack trace	15%	17%	13%	19%	35%
Steps to reproduce	3%	10%	33%	40%	14%
Test cases, test scripts	3%	9%	40%	26%	23%
User input	2%	10%	31%	45%	12%

Least missing or incorrect pieces of information, based on the "Never or very seldom" rating, were *reporter's contact information* (47%), *bug title* (32%), and *product information* (15%). Out of these three, *product information* was missing

more than others, as its median answer option was "Sometimes" and with others, the median answer option was "Quite seldom." *Reporter's contact information* and *bug title* are both easily identified by the reporter and can be seen as a de facto standard of any kind of communication. For example, when sending email both title and sender's contact information are generally included in the message.

It seems that defect reporting is not flawless, as many useful items such as *steps to reproduce, stack trace,* and *configuration of the application*, were not included in the report or were incorrect quite often according to over 50% of the participants. *Test cases and test scripts, steps to reproduce, user input, expected behavior*, and *configuration of the application* were rated to be missing or incorrect "Sometimes" or more often by over 80%. It is possible that a reporter does not include the information, because he knows exactly which items are related to the defect. In any case it seems that there are problems with the information content of defect reports. If some items would be collected automatically to the reports, then they would not be missing and quite certainly not be incorrect.

C. Automatic Information Collection

Developers rated items based on how easily items could be collected automatically. The items most rated to be **very easily collectible** were: *product information* (45%), *reporter's contact information* (35%), and *hardware context* (32%). Collecting them can be seen as a rather straightforward process as all of them are usually available from desktop applications. Items that were **the most difficult to collect** were: *expected behavior* (45%), *bug title* (28%), and *severity of the bug* (25%). These items are usually all defined by the reporter, even in automatic defect reporting applications. Some developers thought that these could be collected automatically too, but we propose that the quality of information would suffer from automatic collection. Participants could also answer with neutral "I don't know" option. This option was used by 12–42% of respondents depending on the item in question (Table V).

No common understanding was found among the participants regarding the automatic collection of several items. This could mean that while some items are generally easy to collect, other items can be more difficult to collect in a specific context. Participants were more inclined to rate information as easy to collect, perhaps because it is easier to determine if it is easy to collect the information than to determine if it is difficult. The answers could also have been biased by *social desirability bias* [27]. Either way, items such as *expected behavior* can be said to be difficult to collect even though 16% of developers answered the other way.

D. Combinations

Defect reporting could be improved by only inspecting which items are the most useful for fixing defects. However, the most useful item for defect fixing was *steps to reproduce* which is difficult to collect automatically (Figure 1). Therefore we also inspected the combinations of different factors to analyze which items should be collected automatically into defect reports.

TABLE V
AUTOMATIC COLLECTION OF ITEMS

Item	Very Hard	Quite Hard	I Don't Know	Quite Easy	Very Easy
Bug title	28%	17%	23%	15%	17%
Component / module	2%	23%	12%	44%	19%
Configuration	0%	28%	18%	49%	5%
Error reports	0%	21%	31%	33%	15%
Expected behavior	45%	12%	27%	10%	6%
Hardware context	0%	4%	16%	48%	32%
Observed behavior	11%	30%	35%	23%	2%
Operating data	3%	23%	26%	39%	9%
Part of the application	1%	25%	24%	31%	19%
Product information	2%	0%	17%	36%	45%
Contact information	2%	2%	28%	33%	35%
Screenshots	11%	13%	36%	31%	9%
Severity of the bug	25%	15%	42%	12%	5%
Software context	0%	7%	19%	45%	29%
Stack trace	0%	17%	29%	29%	25%
Steps to reproduce	14%	39%	25%	19%	3%
Test cases, test scripts	20%	29%	34%	14%	3%
User input	10%	33%	31%	24%	2%

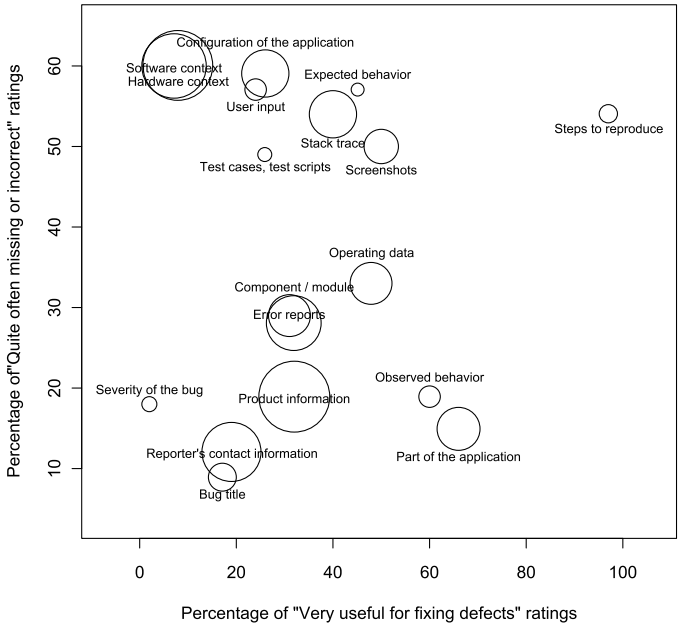

Fig. 1. Summary of The Results. Size of circles is determined by difficulty of automatic collection (larger circles are easier).

We calculated four combinations: critical, solution, booster, and essential. For example, we calculated which items were most often rated useful and missing or incorrect to see which items were the most critical (Table VI). The scores were calculated by first examining if the ratings by an individual developer were at least "Quite useful," "Quite often," or "Quite easy," depending on which combination was calculated (Table VI). Then the number of satisfying answers was com-

pared to the total number of selections on the Page 2 of the questionnaire. **The most critical items** were *steps to reproduce, expected behavior,* and *user input*. Collection of these items should be made with manually, since they are generally difficult to collect automatically. *Hardware context, software context,* and *configuration of the application* could be collected automatically and should be, because they cause problems. Collecting *part of the application, product information,* and *stack trace* automatically would reduce the manual effort to report defects, thus they would boost defect reporting. *Stack trace, configuration of the application,* and *component / module* could be collected automatically to decrease problems with defect reporting in general, although the percentages were lower compared to other combinations (14–21%).

TABLE VI
COMBINATIONS

	Useful	Missing or Incorrect	Automatic Collection	High Scores
Critical	✓	✓		steps to reproduce (54%) expected behavior (53%) user input (43%)
Solution		✓	✓	hardware context (44%) software context (43%) configuration (31%)
Booster	✓		✓	part of the application (47%) product information (47%) stack trace (46%)
Essential	✓	✓	✓	stack trace (21%) configuration (18%) component / module (14%)

V. DISCUSSION

In this section, we answer our research questions, compare our results with previous research and discuss threats to validity.

A. What Defect Information Do Developers Consider Useful for Fixing Defects?

The participants of the survey selected pieces of information they considered useful for fixing defects. After that, they rated these items according to how useful they are. Most developers considered *steps to reproduce, screenshots*, and *part of the application* being useful for fixing software defects. *Observed behavior* was not selected to be useful by as many as the top three, but it was rated to be highly useful by those who selected it. Several other items were selected multiple times and can be considered quite useful for fixing defects (Table III).

Zimmermann et al. [1] conducted a survey similar to ours. Our results and their results relate to each other in three different ways. First, our results confirm some of their results. Second, some of our results differ from their results. Third, we extend their results with knowledge about additional pieces of defect information.

Some of their results were similar to ours, since *steps to reproduce, observed behavior*, and *expected behavior* stood up being important for fixing defects in both surveys. *Steps to reproduce* was clearly the most important item in both our study and their study. However, in our study, *screenshots* was selected by 70 out of 74 developers, it being the second most selected item, but in the study by Zimmermann et al. only 26% selected *screenshots* as one of the three most important pieces of information. In contrast, *test cases* and *stack traces* were rated to the top three by open source developers but quite low in our research context.

With *screenshots*, it must be considered if the application in question has graphical elements that could be reported with a *screenshot*. Since in our study almost all developers selected *screenshots* to be useful for fixing defects, it shows that all of them had something to do with graphical elements. However, in the study by Zimmermann et al., the survey was conducted for a population of developers for Apache web server, Mozilla web browser, and Eclipse IDE (Integrated Development Environment). The Apache developers had not used *screenshots* for defect fixing as often as the Mozilla developers and the Eclipse developers. Further inspection of the results shows, that when rated by Eclipse developers, *screenshots* were also in the top three most important items. Mozilla developers had used *screenshots* for defect fixing as often as Eclipse developers, but did not select *screenshots* as one of the top three important items as frequently. However, we think that *screenshots* are important also for Mozilla developers. Because the developers were only allowed to select three items as the most important, the results can be misinterpreted to mean that Mozilla developers did not consider *screenshots* as important.

With *test cases* and *stack traces*, we think that difference in the results comes from the differences between open source and industrial defect reporting cultures. In the open source community, defect reporters can be developers themselves and they can have advanced knowledge about defect fixing. In the other hand in the industry defects are reported by different stakeholders [2] and their technical knowledge can be limited.

Zimmermann et al. used a preselected list of useful pieces of information for fixing defects. Compared to the list used in our survey, they did not list *part of the application*, *configuration of the application*, *operating data*, and *user input*. Out of these items, *part of the application*, *configuration of the application*, and *operating data* were selected by 82–92% of developers (Table III) and thus can be seen as useful for fixing defects. Hence, our research extends the knowledge from the previous research.

As it was mentioned in the results, the developers were not unanimous in their responses. Developers working with different parts of an application or in different operation environments consider different items useful, which is also summarized in a comment: *"Different kind of bugs require different input for fixing."* Thus, there is defect information that is useful with every defect, such as *steps to reproduce, observed behavior*, and *expected behavior*. Some useful information depends on the context, such as *operating data, user input*, and *Internet page address*. The results from the survey tell us what information is useful for fixing defects,

but items rated lower can still be useful in some cases. Similar caution was made by Zimmermann et al. [1]: *"Items with low importance in our survey are not totally irrelevant because they still might be needed to understand, reproduce, or triage bugs."*

B. Has the Useful Information Been Incorrect or Missing in Defect Reports?

Developers rated pieces of defect information based on how often they had been missing or incorrect in defect reports. Only items they considered useful were rated. *Hardware context, software context*, and *stack trace* had been incorrect or missing in defect reports most often. When looking the items selected to be very useful in the previous section on the survey, we see that *steps to reproduce* and *screenshots* are rated to be missing or incorrect quite often. All items, except *reporter's contact information* and *bug title*, were rated to be missing or incorrect at least sometimes by the majority of the participants. Based on these observations we argue that quality of defect reports varies across the reports in the case companies, and useful items are not reported systematically enough.

Regarding the defect report quality, our survey results rest on developers' experience, not on examination of defect repositories. Multiple studies of the quality of defect reports have been made by classifying reports in open bug repositories, such as BugZilla. Anvik et al. [28] proposed that 39% of the defect reports in Eclipse-project and 56% in Firefox-project do not contribute to improving the project, as those reports were not marked as *open* or *fixed* in the bug tracking system. Schugerl et al. [29] studied the defect reports of ArgoUML-project and calculated with a similar procedure as of Anvik et al., that only 67% of the reports contribute the project. Based on prior work and our results, it seems that defect reporting is far from ideal.

Zimmermann et al. [1] also surveyed defect reporters, in addition to software developers. Their sample of software developers thought that most problems in defect reports were caused by incomplete information. Errors in *steps to reproduce* and *observed behavior* caused some of the most severe problems, which matches our results in which these items were considered very useful for fixing defects. Another similarity between our study and that of Zimmermann et al. is that their sample of defect reporters did not frequently report *hardware context* and *stack traces*. In our results, these items were the most often missing or incorrect. Other results by Zimmermann et al. provide us two causes why there are problems with defect reporting. First, reporters in the study by Zimmermann et al. did not consider *hardware context* relevant to fixing defects and thus reporting them would be only waste of time. Second, *steps to reproduce* and *stack traces* were considered relevant, but they were considered difficult to report. These discoveries could be the causes why certain items are not reported appropriately.

C. Which Information Should Automatically Be Collected into Defect Reports?

Developers rated pieces of defect information based on how easily they could automatically be collected into defect reports. The items rated to be most easily collectible were *product information, reporter's contact information*, and *hardware context*. Developers also thought that the most difficult to collect are *expected behavior, severity of the bug*, and *test cases, test scripts*. The quantitative results here do not show how to collect the information; instead, the results reflect developers' personal thoughts about automatic defect reporting.

To analyze which items should be collected automatically, we calculated four combinations of the answers. First, we calculated the most critical pieces of information. These items were the most useful and the most missing or incorrect. The most critical items were *steps to reproduce, expected behavior*, and *user input*. However, these items are considered difficult to collect automatically. Thus, we see that collecting these items can be most easily enhanced by training defect reporters. Second, items that generally are missing or incorrect, but could be collected automatically were called *solutions*. *Hardware context, software context*, and *configuration of the application* caused most problems that could be solved. These items should be the first ones to collect automatically, since they cause most problems according to our survey. Third, items that were the most useful and the easiest to collect automatically were called *boosters*. Top three booster items were *part of the application, product information*, and *stack trace*. Collecting these items should be the second priority because at least they reduce the defect reporting workload. Finally, items that were useful, problematic, and automatically collectible were seen as *essential*. *Stack trace, configuration of the application*, and *component/module* were seen as the most essential items. However, only 14–21% of the developers saw these items as we defined essential. Thus, it seems that there is no single item that would solve the problems related to defect reporting.

D. Threats to Validity

Our research was conducted as a survey research to have a general understanding about the state of defect reporting in our target companies. Survey research does not directly measure different qualities. Survey participants had to have a common understanding on the questions and the answering options in the survey. Particularly the list of defect information had to apply to all domains of software development as much as possible. To implement automatic defect reporting, one must define more strictly how to collect part of the application, for example. The comments from the survey reveal that some participants were confused about the ambiguous items: *"Bugs can be so different (from crashes to wrong UI texts, from wrong software behaviour to usability issues, etc.) that I felt difficult to answer to these generic questions. Different kind of bugs require different input for fixing."*

To make conclusions about a certain population, we must study the whole population, or more feasibly a random sample of the population. Our sample was based on earlier research

cooperation. While there were developers from many software projects in the population, it was not systematically random, thus the results might be different if the study is replicated with different companies. Since the results were handled as one population, we assumed that the population was homogeneous and the measured qualities could be applied to a larger population. This might not be the case with all items in our preselected list. Our results do not show the differences between distinct software projects, but present a general view which could be deepened in the future.

We think that our results can be generalized to some extent. The results did not vary remarkably among our target companies. Thus, the results should be at least somewhat similar with other areas of software development. *Steps to reproduce* was unanimously the most important item for fixing defects and this verifies the result of an earlier study in the context of open source software development [1].

VI. Conclusion and Future Work

To understand the problems related to defect reporting, we conducted a survey in the context of industrial software development about defect reporting. We reproduced the survey of the previous study [1] by Zimmermann et al. which was conducted in the context of open source software development. Our results strengthen, present differences and extend the previous results. Our results verify that *steps to reproduce* and *observed behavior* are highly important defect information. We found differences in comparison to original study as *stack trace* and *test cases* were seen less useful in our study. On the other hand, *screenshots* were seen more important in our study. Finally, we extend the original study as we found that *part of the application*, *configuration of the application*, and *operating data* are important, but they were not surveyed in the original work.

In this paper, we also studied defect information from three viewpoints: usefulness, correctness and feasibility for automatic collection. When combining these three dimensions, we found that there are no defect data items that would score high in all three dimensions. If such items existed, they would be visible in the top right corner with a big circle in (Figure 1). However, there are items that are often missing and easy to automate, but not very useful for defect fixing. Furthermore there are items that are very useful and easy to automate, but not often missing or incorrect. Nevertheless, the automatic collection of such items could be beneficial as the needed defect information for fixing varies between different types of defects, and as defect data can be used for other purposes than defect fixing such as resource allocation, e.g. targeting testing resources to a particular software context where many defects are found.

In the future, people implementing automatic defect reporting or defect reporting systems in general should utilize these results. Different processes should be determined, such as determining the useful information for a distinct software project and practical implementation of the reporting system.

Acknowledgment

We thank Ville Heikkilä and Rahul Premraj for improving this paper. We appreciate the help of Timo Lehtinen and Jari Vanhanen gave us in constructing the questionnaire. The survey conducted in this paper would have not been possible without the help of contact people at the participating companies and of course the developers who completed the survey. We acknowledge their effort.

References

[1] T. Zimmermann, R. Premraj, N. Bettenburg, J. Sascha, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Trans. Software Eng.*, vol. 36, no. 5, pp. 618–643, 2010.

[2] M. Mäntylä, J. Iivonen, and J. Itkonen, "Who tested my software? testing as an organizationally cross-cutting activity," minor revision submitted for Software Quality Journal.

[3] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood, "Replication of experimental results in software engineering," International Software Engineering Research Network (ISERN) Technical Report ISERN- 96-10, University of Strathclyde, Tech. Rep., 1996.

[4] N. Juristo and S. Vegas, "Using differences among replications of software engineering experiments to gain knowledge," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 356–366.

[5] F. Shull, J. Carver, S. Vegas, and N. Juristo, "The role of replications in Empirical Software Engineering," *Empirical Software Engineering*, vol. 13, no. 2, pp. 211–218, 2008.

[6] B. Kitchenham and S. Pfleeger, "Principles of survey research part 4: questionnaire evaluation," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 3, pp. 20–23, 2002.

[7] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 401–404.

[8] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[9] D. Sjoberg, T. Dyba, and M. Jorgensen, "The future of empirical methods in software engineering research," *Future of Software Engineering*, pp. 358–378, 2007.

[10] M. Mäntylä, C. Lassenius, and J. Vanhanen, "Rethinking replication in software engineering: Can we see the forest for the trees?" in *ICSE workshop RESER*.

[11] O. S. Gómez, N. Juristo, and S. Vegas, "Replications types in experimental disciplines," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 3:1–3:10. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852790

[12] S. Tatham, "How to report bugs effectively," September 2008. [Online]. Available: http://www.chiark.greenend.org.uk/~sgtatham/bugs.html

[13] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 298–308.

[14] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010, pp. 301–310.

[15] R. Merkel and T. Kanij, "Does the Individual Matter in Software Testing?" Tech. Rep., 2010.

[16] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, "Debugging in the (very) large: ten years of implementation and experience," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 103–116. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629586

[17] S. Artzi, S. Kim, and M. Ernst, "Recrashj: A tool for capturing and reproducing program crashes in deployed applications," 2009, pp. 295–296.

[18] M. Castro, M. Costa, and J.-P. Martin, "Better bug reporting with better privacy," *SIGPLAN Not.*, vol. 43, pp. 319–328, March 2008. [Online]. Available: http://doi.acm.org/10.1145/1353536.1346322

[19] C. Zamfir and G. Candea, "Low-Overhead Bug Fingerprinting for Fast Debugging," in *Runtime Verification*, ser. Lecture Notes in Computer Science, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Rosu, O. Sokolsky, and N. Tillmann, Eds., vol. 6418. Springer Berlin / Heidelberg, 2010, pp. 460–468.

[20] Limesurvey. http://www.limesurvey.org/. Last accessed: 28.1.2011.

[21] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting On-line Surveys in Software Engineering," in *International Symposium on Empirical Software Engineering*, 2003, pp. 80–88.

[22] H. Coolican, *Research methods and statistics in psychology*. Hodder & Stoughton London, 1999.

[23] W. Foddy, *Constructing questions for interviews and questionnaires: theory and practice in social research*. Cambridge Univ Pr, 1994.

[24] O. S. Gómez, N. Juristo, and S. Vegas, "Replication, reproduction and re-analysis: Three ways for verifying experimental findings," in *ICSE workshop RESER*, 2010.

[25] E. Tsang and K. Kwan, "Replication and theory development in organizational science: A critical realist perspective," *Academy of Management Review*, vol. 24, no. 4, pp. 759–780, 1999.

[26] M. Mäntylä and J. Vanhanen, "Software deployment activities and challenges an industrial case study of four software product companies," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering, CSMR*, 2011.

[27] R. Fisher, "Social desirability bias and the validity of indirect questioning," *Journal of Consumer Research*, vol. 20, no. 2, pp. 303–315, 1993.

[28] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, ser. eclipse '05. New York, NY, USA: ACM, 2005, pp. 35–39. [Online]. Available: http://doi.acm.org/10.1145/1117696.1117704

[29] P. Schugerl, J. Rilling, and P. Charland, "Mining bug repositories–a quality assessment," in *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, 2008, pp. 1105 –1110.